seance_2020-11-24_com

November 26, 2020

1 Séance du 24/11/2020

1.1 Listes et boucle "for''

Nouveau type de données: le type "liste": une collection de valeurs, ordonnées.

Les variables de ce type sont définies avec les crochets carrés, les valeurs sont séparées par des virgules.

La variable `lst` déclarée ci dessous contient donc 5 éléments, tous des entiers.

Les crochets servent aussi à faire référence à un élément particulier de la liste: `lst`[i] désigne l'élément numéroté i dans la liste `lst`.

La numérotation des éléments commence toujours à zéro.

La liste lst content donc 5 éléments numérotés de 0 à 4.

[6]:
```
lst = [1,2,3,4,5]
print(lst[3])
```

```
4
```

On peut aussi extraire d'une liste une sous-liste (on parle de tranche, ou *slice*). Le caractère : est utilisé dans ce cas. Voir les exemples et détails dans le cadre pertinent du mémento.

La tranche [*i*:*j*] commence à l'élément numéroté *i* et se termine à l'élément numéroté *j-1*.

[7]:
```
print(lst[3:5])
```

```
[4, 5]
```

Définition des variables qui vont servir d'exemples pour la suite. A noter: - une liste données peut contenir des données de n'importe quel type - en particulier une liste peut contenir une autyre liste etc.

[8]:
```
toto = [3, 7, 9, 1, 0, 8, 8, 8]
titi = [9, False, 8.34, True, 22, 1e-3, 'a', -1]
tutu = [[0,1], ['a', 'b', 'c'], 78.5]
vecteur = [0]*100
lv = []
```

Pour toutes les variables de type liste, on peut utiliser la fonction `len()` (pour *length*, longueur) qui donne le nombre d'éléments de la liste.

On voit ci dessous que la liste `tutu` contient 3 éléments (dont les deux premiers sont aussi des listes). On voit aussi que le deuxième élément de `tutu` (noté `tutu[1]`) est une liste, on peut faire référence au troisième élément de cette liste, avec la notation `tutu[1][2]`.

[9]:
```
print(len(tutu))
print(tutu[1])
print(tutu[1][2])
```

```
3
['a', 'b', 'c']
c
```

Un exemple de tranche: on peut ne pas spécifier le début de la tranche et dans ce cas ça commence à 0.

[10]:
```
print(vecteur[:10])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Modification des éléments d'une liste: si (et seulement si) une liste `l` contient déjà un élément numéroté *i*, alors on peut faire une affectation de la forme `l[i] =`*valeur* pour remplacer le i-ième élément par cette nouvelle valeur. Si la liste `l` contient moins de *i+1* éléments, cette affectation déclenchera une erreur.

[11]:
```
vecteur[3] = 78
```

[12]:
```
print(vecteur[:10])
```

```
[0, 0, 0, 78, 0, 0, 0, 0, 0, 0]
```

On a déjà vu diverses fonctions qui permettent des "conversions de type" (fonctions `int()`, `float()`, *etc.*). Il existe aussi une fonction `list()` qui convertit en liste. Par exemple une chaîne de caractères peut facilement être convertie en liste:

[11]:
```
s = list("Mon beau navire ô ma mémoire")
print(s[5])
print(s)
```

```
e
['M', 'o', 'n', ' ', 'b', 'e', 'a', 'u', ' ', 'n', 'a', 'v', 'i', 'r', 'e', ' ',
'ô', ' ', 'm', 'a', ' ', 'm', 'é', 'm', 'o', 'i', 'r', 'e']
```

Puisque les différents éléments d'une liste sont *indexés* par un entier (de 0 à len()-1), on peut utiliser une boucle avec compteur pour manipuler successivement les éléments d'une liste. Ici par exemple on va imprimer un par un les éléments de la liste `toto`:

[12]:
```
i = 0
while i < 8:
```

```
    print(toto[i])
    i += 1
```

```
3
7
9
1
0
8
8
8
```

On rappelle que si on tente de faire référence à un élément qui n'existe pas, on déclenche une erreur:

[13]:
```
print(toto[9])
```

```
␣
↪---------------------------------------------------------------------------

       IndexError                                Traceback (most recent call␣
↪last)

       <ipython-input-13-922b1c903ec0> in <module>
   ----> 1 print(toto[9])


       IndexError: list index out of range
```

Dans la boucle précédente, la taille de la liste (8) était utilisée directement ("en dur"). Il est bien préférable d'utiliser la longueur de la liste pour éviter de créer des incohérences.

[14]:
```
i = 0
while i < len(tutu):
    print(tutu[i])
    i += 1
```

```
[0, 1]
['a', 'b', 'c']
78.5
```

On peut même définir une fonction générique qui affiche les éléments d'une liste:

[16]:
```
def affiche_liste(l):
    i = 0
    while i < len(l):
        print(l[i], end=' ; ')
        i += 1
```

[17]:
```
affiche_liste(titi)
```

```
9 ; False ; 8.34 ; True ; 22 ; 0.001 ; a ; -1 ;
```

Exercice: afficher le premier, le troisième et le cinquième élément de la liste toto

[19]:
```
# Réponse la plus simple:
print(toto[0], toto[2], toto[4])
# Réponse en utilisant la tranche (et en particulier le pas)
print(toto[:6:2])
```

```
3 9 0
[3, 9, 0]
```

Exercice: afficher les éléments d'indice pair de la liste titi

[21]:
```
i = 0
while i < len(titi):
  print(titi[i])
  i = i + 2
```

```
9
8.34
22
a
```

Variante: on fait varier i de 1 à len(titi)-1, et on utilise un test pour choisir les cas pertinents.

[22]:
```
i = 0
while i < len(titi):
    if i % 2 == 0:
        print(titi[i])
    i = i + 1
```

```
9
8.34
22
a
```

Discussion: Avantages et inconvénients des deux variantes: - la première version est plus rapide: voir ci-dessous les deux variantes avec une variable permettant de compter le nombre de tours ; - la seconde version est plus généralisable (et potentiellement plus lisible)

[25]:
```
# Première variante, avec compteur
i = 0
c = 0
while i < len(titi):
    c += 1
    print(titi[i])
    i = i + 2
```

```
print("%d tours" % c)
```

```
9
8.34
22
a
4 tours
```

[24]:
```
# Seconde variante, avec compteur
i = 0
c = 0
while i < len(titi):
    if i % 2 == 0:
        print(titi[i])
    c += 1
    i = i + 1
print("%d tours" % c)
```

```
9
8.34
22
a
8 tours
```

Un exemple de “remplissage” d’une liste par une boucle d’interaction:

[27]:
```
tirages = [0]*6
i = 0
while i < len(tirages):
    tirages[i] = int(input("Entrez le %d ieme nombre " % (i+1)))
    i += 1
print(tirages)
```

```
Entrez le 1 ieme nombre 5
Entrez le 2 ieme nombre 6
Entrez le 3 ieme nombre 3
Entrez le 4 ieme nombre 89
Entrez le 5 ieme nombre 45
Entrez le 6 ieme nombre 6
[5, 6, 3, 89, 45, 6]
```

Exercice: à partir d’une liste de 10 entiers initialisée à 0, créer une liste comprenant tous les entiers de 1 à 10 dans l’ordre

[28]:
```
lsti = [0]*10
i = 0
while i < len(lsti):
    lsti[i] = i+1
    i += 1
```

```
print(lsti)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercice: Dans une liste d'entiers, remplacer tous les entiers impairs par leur double pour avoir une liste de nombre pairs

[29]:
```
lsti = [3, 7, 9, 1, 0, 8, 8, 8]
i = 0
while i < len(lsti):
    if lsti[i] % 2 != 0:
        lsti[i] = lsti[i] * 2
    i += 1
print(lsti)
```

```
[6, 14, 18, 2, 0, 8, 8, 8]
```

1.1.1 Boucle 'for'

La structure de contrôle 'for x in l' permet de réaliser une boucle dans laquelle la variable x prend successivement (à chaque tour) la valeur du prochain élément de la liste l:

[30]:
```
for t in lsti:
    print(t)
```

```
6
14
18
2
0
8
8
8
```

Exemple: afficher les éléments de la liste toto qui sont pairs:

[31]:
```
for x in toto:
    if x % 2 == 0:
        print(x)
```

```
0
8
8
8
```

Exemple: utilisation du mot-clé `in` pour définir un booléen (`in` doit être lu comme *appartient à*):

[33]:
```
s = list("Mon beau navire ô ma mémoire")
c = 0
for l in s:
```

```
    if l in ['a', 'e', 'i', 'o', 'u']:
        c += 1
print('le mot contient %d voyelles' % c)
```

```
le mot contient 11 voyelles
```

Fonction très couramment utilisée avec la commande `for`: `range(i)` crée une [sorte de] liste d'entiers, de 0 à *i-1*. `range(i,j)` crée une liste d'entiers de *i* à *j-1*.

[35]:
```
for x in range(8):
    print(x)
```

```
0
1
2
3
4
5
6
7
```

[36]:
```
f = range(3,8)
for x in f:
    print(x)
```

```
3
4
5
6
7
```

Pour illustrer l'intérêt de l'instruction `for`, on peut comparer les deux variantes ci-dessous, qui réalisent la même opération. L'avantage de la version avec `for` est qu'elle est plus compacte et plus lisible, et moins dangereuse, car on n'a pas besoin de "gérér" explicitement la variable `c` (on a vu qu'il était possible d'oublier l'initiatilisation, ou l'incrémentation, ce qui cause à chaque fois un comportement non souhaité).

[41]:
```
c = 0
while c < 10:
    print(c, end=' ; ')
    c += 1
```

```
0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ;
```

[42]:
```
for c in range(10):
    print(c, end=' ; ')
```

```
0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ;
```